Inner class means one class which is a member of another class. There are basically four types of inner classes in java.

1) Nested Inner class  
2) Method Local inner classes  
3) Anonymous inner classes  
4) Static nested classes

**Nested Inner class** can access any private instance variable of outer class. Like any other instance variable, we can have access modifier private, protected, public and default modifier.  
Like class, interface can also be nested and can have access specifiers.

class Outer {

   // Simple nested inner class

   class Inner {

      public void show() {

           System.out.println("In a nested class method");

      }

   }

}

class Main {

   public static void main(String[] args) {

       Outer.Inner in = new Outer().new Inner();

       in.show();

   }

}

As a side note, we can’t have static method in a nested inner class because an inner class is implicitly associated with an object of its outer class so it cannot define any static method for itself. For example the following program doesn’t compile

class Outer {

   void outerMethod() {

      System.out.println("inside outerMethod");

   }

   class Inner {

      public static void main(String[] args){

         System.out.println("inside inner class Method");

      }

   }

}

An interface can also be nested and nested interfaces have some interesting properties

**Method Local inner classes**  
Inner class can be declared within a method of an outer class. In the following example, Inner is an inner class in outerMethod().

|  |
| --- |
| class Outer {      void outerMethod() {          System.out.println("inside outerMethod");          // Inner class is local to outerMethod()          class Inner {              void innerMethod() {                  System.out.println("inside innerMethod");              }          }          Inner y = new Inner();          y.innerMethod();      }  }  class MethodDemo {      public static void main(String[] args) {          Outer x = new Outer();          x.outerMethod();      }  } |

Method Local inner classes can’t use local variable of outer method until that local variable is not declared as final. For example, the following code generates compiler error (Note that x is not final in outerMethod() and innerMethod() tries to access it)

|  |
| --- |
| class Outer {     void outerMethod() {        int x = 98;        System.out.println("inside outerMethod");        class Inner {           void innerMethod() {              System.out.println("x= "+x);           }        }        Inner y = new Inner();        y.innerMethod();     }  }  class MethodLocalVariableDemo {     public static void main(String[] args) {        Outer x=new Outer();        x.outerMethod();     }  } |

Output:

local variable x is accessed from within inner class;

needs to be declared final

**Note :**Local inner class cannot access non-final local variable till JDK 1.7. Since JDK 1.8, it is possible to access the non-final local variable in method local inner class

The main reason we need to declare a local variable as a final is that local variable lives on stack till method is on the stack but there might be a case the object of inner class still lives on the heap.  
Method local inner class can’t be marked as private, protected, static and transient but can be marked as abstract and final, but not both at the same time.

**Static nested classes**  
Static nested classes are not technically an inner class. They are like a static member of outer class.

|  |
| --- |
| class Outer {     private static void outerMethod() {       System.out.println("inside outerMethod");     }       // A static inner class     static class Inner {       public static void main(String[] args) {          System.out.println("inside inner class Method");          outerMethod();       }     }    } |

**Anonymous inner classes**  
Anonymous inner classes are declared without any name at all. They are created in two ways.  
***a)****As subclass of specified type*

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| class Demo {     void show() {        System.out.println("i am in show method of super class");     }  }  class Flavor1Demo {       //  An anonymous class with Demo as base class     static Demo d = new Demo() {         void show() {             super.show();             System.out.println("i am in Flavor1Demo class");         }     };     public static void main(String[] args){         d.show();     }  } |

Output

i am in show method of super class

i am in Flavor1Demo class

In the above code, we have two class Demo and Flavor1Demo. Here demo act as super class and anonymous class acts as a subclass, both classes have a method show(). In anonymous class show() method is overridden.

***a)****As implementer of the specified interface*

|  |
| --- |
| class Flavor2Demo {        // An anonymous class that implements Hello interface      static Hello h = new Hello() {          public void show() {              System.out.println("i am in anonymous class");          }      };        public static void main(String[] args) {          h.show();      }  }    interface Hello {      void show();  } |

Output:

i am in anonymous class

In above code we create an object of anonymous inner class but this anonymous inner class is an implementer of the interface Hello. Any anonymous inner class can implement only one interface at one time. It can either extend a class or implement interface at a time.